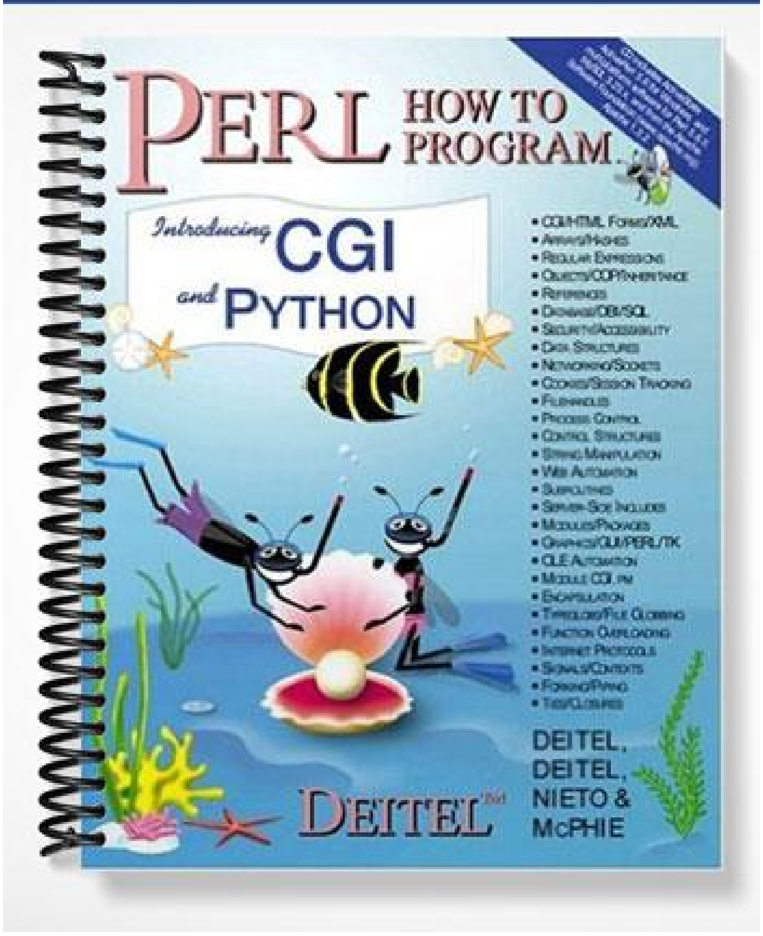
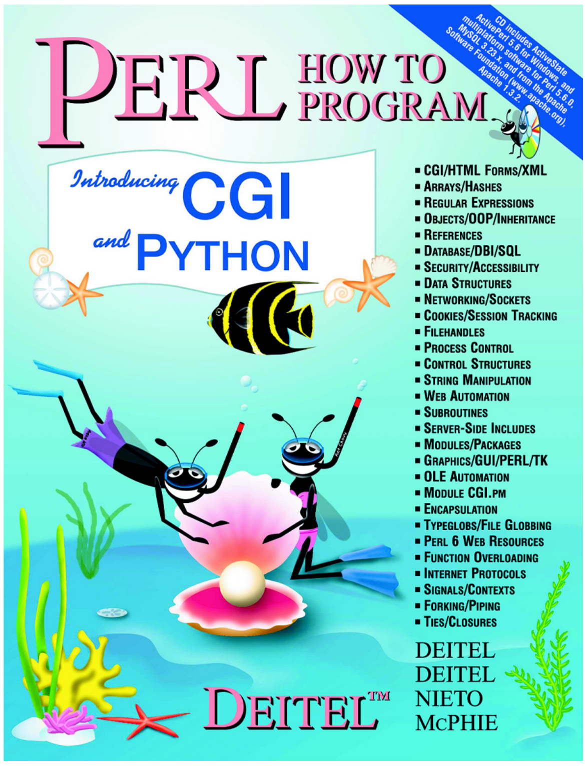
SOLUTIONS MANUAL



Instructor's Manual

for

Perl
How to Program



Harvey M. Deitel
Paul J. Deitel
Tem R. Nieto
David C. McPhie

Contents

Preface

Thank you for considering and/or adopting our text *Perl How to Program*. If you have not read the preface to *Perl How to Program*, please do so. The preface contains a careful walkthrough of the book's key features, including the case studies.

We have worked hard to produce a textbook and ancillaries that we hope you and your students will find valuable. The following ancillary resources are available:

- *Perl How to Program's examples* are included on the CD-ROM in the back of the textbook. This helps instructors prepare lectures faster and helps students master Internet programming. The examples are also available for download at **`www.deitel.com`** and **`www.prenhall.com/deitel`**. When extracting the source code from the ZIP file, you must use a ZIP-file reader such as WinZip (**`www.winzip.com`**) or PKZIP (**`www.pkware.com`**) that understands directories. The file should be extracted into a separate directory (e.g., **`perlhtp_examples`**).
- The CD-ROM included with *Perl How to Program* contains a variety of software, including [***].
- *This Perl How to Program Instructor's Manual* on CD contains answers to most of the exercises in the textbook. The programs are separated into directories by chapter and exercise number.
- *Companion Web site* (**`www.prenhall.com/deitel`**) provides instructor and student resources. Instructor resources include textbook appendices (e.g., Appendix A, "HTML Special Characters") and a syllabus manager for lesson planning. Student resources include chapter objectives, true/false questions, chapter highlights, reference materials and a message board.
- Customizable *Powerpoint Instructor Lecture Notes*, with many complete features including source code, and key discussion points for each program and major illustration. These lecture notes are available for instructors and students at no charge at our Web site **`www.deitel.com`** and **`www.prenhall.com/deitel`**.

We would sincerely appreciate your questions, comments, criticisms and corrections addressed to us at:

`deitel@deitel.com`

We will respond immediately. Please read the latest copy of the *Deitel Buzz* (published every April and November) for information on forthcoming Deitel publications, ancillaries, product options and ordering information. To receive the *Deitel Buzz*, please contact Jennie Burger (**`jennie_burger@prenhall.com`**).

Watch our Deitel & Associates, Inc. Web site (**`www.deitel.com`**) and our Prentice Hall Web site (**`www.prenhall.com/deitel`**) for the latest publication updates.

We would like to thank the extraordinary team of publishing professionals at Prentice Hall who made *Perl How to Program* and its ancillaries possible. Our Computer Science editor, Petra Recter, worked closely with us to ensure the timely availability and professional quality of these ancillaries.

We would also like to thank Cheryl Yaeger and Matt Kowalewski, both of Deitel & Associates, Inc., for their assistance in preparing this Instructor's Manual.

Harvey M. Deitel
Paul J. Deitel
Tem R. Nieto
David C. McPhie

1

Introduction to Computers, the Internet and the World Wide Web

1.2 [CD] Categorize each of the following items as either hardware or software:

a) CPU

ANS: hardware.

b) ALU

ANS: hardware.

c) input unit

ANS: hardware.

d) a word processor program

ANS: software.

e) Perl modules

ANS: software.

1.3 [CD] Why might you want to write a program in a machine-independent language instead of a machine-dependent language? Why might a machine-dependent language be more appropriate for writing certain types of programs?

ANS: Machine independent languages are useful for writing programs to be executed on multiple computer platforms. Machine dependent languages are appropriate for writing programs to be executed on a single platform. Machine dependent languages tend to exploit the efficiencies of a particular machine.

1.4 [CD] Translator programs such as assemblers and compilers convert programs from one language (referred to as the *source* language) to another language (referred to as the *object* language). Determine which of the following statements are *true* and which are *false*:

a) A compiler translates high-level language programs into object language.

ANS: True.

b) An assembler translates source language programs into machine language programs

ANS: True.

c) A compiler converts source language programs into object language programs.

ANS: False.

d) High-level languages are generally machine-dependent.

ANS: False.

e) A machine language program requires translation before it can be run on a computer.
ANS: False.

1.5 Fill in the blanks in each of the following statements:

a) Devices from which users access timesharing computer systems are usually called __________.
ANS: terminals.

b) A computer program that converts assembly language programs to machine language programs is called __________.
ANS: an assembler.

c) The logical unit of the computer that receives information from outside the computer for use by the computer is called __________.
ANS: The input unit.

d) The process of instructing the computer to solve specific problems is called __________.
ANS: computer programming.

e) What type of computer language uses English-like abbreviations for machine language instructions? __________.
ANS: a high-level language.

f) What are the six logical units of the computer? __________.
ANS: input unit, output unit, memory unit, arithmetic and logical unit, central processing unit, secondary storage unit.

g) Which logical unit of the computer sends information that has already been processed by the computer to various devices so that the information may be used outside the computer? __________.
ANS: The output unit.

h) The general name for a program that converts programs written in a certain computer language into machine language is __________.
ANS: compiler.

i) Which logical unit of the computer retains information? __________.
ANS: memory unit and secondary storage unit.

j) Which logical unit of the computer performs calculations? __________.
ANS: arithmetic and logical unit.

k) Which logical unit of the computer makes logical decisions? __________.
ANS: arithmetic and logical unit

l) The commonly used abbreviation for the computer's control unit is __________.
ANS: CPU.

m) The level of computer language most convenient to the programmer for writing programs quickly and easily is __________.
ANS: high-level language.

n) The most common business-oriented language in wide use today is __________.
ANS: COBOL.

o) The only language that a computer can directly understand is called that computer's __________.
ANS: machine language.

p) Which logical unit of the computer coordinates the activities of all the other logical units? __________.
ANS: central processing unit.

1.6 State whether each of the following is *true* or *false*. If *false*, explain your answer.

a) Machine languages are generally machine dependent.
ANS: True. Machine languages are closely related to the hardware of a particular machine.

b) Timesharing truly runs several users simultaneously on a computer.

ANS: False. Time sharing systems split CPU time amongst several users so that the users appear to be operating simultaneously

c) Like other high-level languages, Perl is generally considered to be machine-independent.

ANS: True. Perl programs can be written on most machines, and in most cases, Perl programs can be written on one machine and run on many machines with few changes or no changes.

2

Introduction to Programming in Perl

2.7 [CD] Write a program that asks the user to enter two numbers and prints the sum, product, difference and quotient of the two numbers.

ANS:

```perl
#!/usr/bin/perl
# Ex. 2.7: Ex02_07.pl
# Calculating sum, product, difference and quotient values.

print( "Enter the first number: " );
$first = <STDIN>;
chomp( $first );

print( "Enter the second number: " );
$second = <STDIN>;
chomp( $second );

print( "$first + $second = ", $first + $second, "\n" );
print( "$first * $second = ", $first * $second, "\n" );
print( "$first - $second = ", $first - $second, "\n" );
print( "$first / $second = ", $first / $second, "\n" );
```

```
Enter the first number: 6
Enter the second number: 3
6 + 3 = 9
6 * 3 = 18
6 - 3 = 3
6 / 3 = 2
```

2.8 [CD] Write a program that asks the user to input the radius of a circle and prints the circle's diameter, circumference and area. Use the value 3.14159 for π. Use the following formulas (*r* is the radius): diameter = 2r, circumference = 2πr, area = πr^2.

ANS:

```
#!/usr/bin/perl
# Ex. 2.8: Ex02_08.pl
# Calculating diameter, circumference and area.

$pi = 3.14159;
print( "Enter the radius: " );
$r = <STDIN>;
chomp( $r );

$diameter = 2 * $r;
$circumference = 2 * $pi * $r;
$area = $pi * $r ** 2;

print( "The diameter of the circle is $diameter.\n" );
print( "The circumference of the circle is $circumference.\n" );
print( "The area of the circle is $area.\n" );
```

```
Enter the radius: 1
The diameter of the circle is 2.
The circumference of the circle is 6.28318.
The area of the circle is 3.14159.
```

2.9 Write a program that asks the user to input one number consisting of five digits, separates the number into its individual digits and prints the fifth digit five times, the fourth digit four times, the third digit three times and so forth.

ANS:

```
#!/usr/bin/perl
# Ex 2.9: Ex02_09.pl
# Printing digit values multiple times.

print( "Enter a five digit number: " );
$number = <STDIN>;
chomp( $number );

$ones = $number % 10;
$tens = ( $number / 10 ) % 10;
$hundreds = ( $number / 100 ) % 10;
$thousands = ( $number / 1000 ) % 10;
$tenthousands = ( $number / 10000 ) % 10;

print( "$tenthousands\n" );
print( $thousands x 2, "\n" );
print( $hundreds x 3, "\n" );
print( $tens x 4, "\n" );
print( $ones x 5, "\n" );
```

```
Enter a five digit number: 12345
1
22
333
4444
55555
```

2.10 Write a program that reads in two integers and determines if either is a multiple of the other.

ANS:

```perl
#!/usr/bin/perl
# Ex 2.10: Ex02_10.pl
# Determining if one integer is a multiple of the other.

print( "Enter the first number: " );
$first = <STDIN>;
chomp( $first );

print( "Enter the second number: " );
$second = <STDIN>;
chomp( $second );

if ( $first % $second == 0 ) {
   print( "$first is a multiple of $second!\n" );
}

if ( $second % $first == 0 ) {
   print( "$second is a multiple of $first!\n" );
}
```

```
Enter the first number: 12
Enter the second number: 4
12 is a multiple of 4!
```

```
Enter the first number: 11
Enter the second number: 2
```

2.11 [CD] Write a program that reads two strings input by the user and prints them in alphabetical order, separated by a space. If the two strings are equal, they should be printed on separate lines.

ANS:

```perl
#!/usr/bin/perl
# Ex 2.11: Ex02_11.pl
# Printing strings in alphabetical order.

print( "Enter the first string: " );
chomp( $first = <STDIN> );
```

```
print( "Enter the second string: " );
chomp( $second = <STDIN> );

if ( $first lt $second ) {
   print( "$first $second\n" );
}

if ( $first eq $second ) {
   print( "$first\n$second\n" );
}

if ( $first gt $second ) {
   print( "$second $first\n" );
}
```

```
Enter the first string: there
Enter the second string: hello
hello there
```

```
Enter the first string: hello
Enter the second string: there
hello there
```

```
Enter the first string: hello
Enter the second string: hello
hello
hello
```

2.12 [CD] One interesting application of computers is drawing graphs and bar charts (sometimes called “histograms”). Write a program that reads five numbers (each between 1 and 30) from user input. For each number read, your program should print a line containing that number of adjacent asterisks. For example, if your program reads the number 7, it should print `*******`.

ANS:

```
#!/usr/bin/perl
# Ex 2.12: Ex02_12.pl
# Printing asterisks representing numerical values.

print( "Enter a number ( 1-30 ): " );
chomp( $number1 = <STDIN> );
print( "Enter a number ( 1-30 ): " );
chomp( $number2 = <STDIN> );
print( "Enter a number ( 1-30 ): " );
chomp( $number3 = <STDIN> );
print( "Enter a number ( 1-30 ): " );
chomp( $number4 = <STDIN> );
print( "Enter a number ( 1-30 ): " );
chomp( $number5 = <STDIN> );
```

```

print( "\nValue\tHistogram\n" );
print( "$number1\t", '*' x $number1, "\n" );
print( "$number2\t", '*' x $number2, "\n" );
print( "$number3\t", '*' x $number3, "\n" );
print( "$number4\t", '*' x $number4, "\n" );
print( "$number5\t", '*' x $number5, "\n" );
```

```
Enter a number ( 1-30 ): 7
Enter a number ( 1-30 ): 29
Enter a number ( 1-30 ): 13
Enter a number ( 1-30 ): 22
Enter a number ( 1-30 ): 3

Value   Histogram
7       *******
29      *****************************
13      *************
22      **********************
3       ***
```

2.13 Write a program that asks the user to input an integer, then prints a hollow square of asterisks. For example, if the user inputs 5, the output should be as follows:

```
*****
*   *
*   *
*   *
*****
```

ANS:

```
#!/usr/bin/perl
# Ex. 2.13: Ex02_13.pl
# Printing a hollow square based on an integer value.

print( "How big should the square be? " );
chomp( $size = <STDIN> );

if ( $size == 1 ) {
   print( "*\n" );
}
else {
   $firstlast = ( '*' x $size ) . "\n";
   $middle = '*' . ( ' ' x ( $size - 2 ) ) . "*\n";
   print( $firstlast, $middle x ( $size - 2 ), $firstlast );
}
```

```
How big should the square be? 3
***
* *
***
```

```
How big should the square be? 4
****
*  *
*  *
****
```

3

Control Structures: Part I

3.10 Drivers are concerned with the gas mileage obtained by their automobiles. One driver has kept track of several tankfuls of gasoline by recording the miles driven and gallons used for each tankful. Develop a program that will receive as input the miles driven and gallons used for each tankful. The program should calculate and display the miles per gallon obtained for each tankful. After processing all input information, the program should calculate and print the combined miles per gallon obtained for all tankfuls. Sample output of the program is as follows:

```
Enter the gallons used ( -1 to end ): 12.8
Enter the miles driven: 287
The miles / gallon for this tank was 22.421875

Enter the gallons used ( -1 to end ): 10.3
Enter the miles driven: 200
The miles / gallon for this tank was 19.417475

Enter the gallons used ( -1 to end ): 5
Enter the miles driven: 120
The miles / gallon for this tank was 24.000000

Enter the gallons used ( -1 to end ): -1

The overall average miles/gallon was 21.601423
```

ANS:

```
#!/usr/bin/perl
# Ex. 3.10: Ex03_10.pl
# Calculating miles per gallon.

print( "Enter the gallons used ( -1 to quit ): " );
```

```
chomp( $gallons = <STDIN> );

while ( $gallons != -1 ) {
   print( "Enter the miles driven: " );
   chomp( $miles = <STDIN> );
   $totalGallons += $gallons;
   $totalMiles += $miles;

   print( "The miles / gallon for this tank was " );
   print( $miles / $gallons, "\n\n" );

   print( "Enter the gallons used ( -1 to quit ): " );
   chomp( $gallons = <STDIN> );
}

unless ( $totalGallons == 0 ) {
   print( "\nThe overall average miles/gallon was " );
   print( $totalMiles / $totalGallons, "\n" );
}
```

```
Enter the gallons used ( -1 to quit ): 15
Enter the miles driven: 200
The miles / gallon for this tank was 13.3333333333333

Enter the gallons used ( -1 to quit ): 17
Enter the miles driven: 34
The miles / gallon for this tank was 2

Enter the gallons used ( -1 to quit ): 10
Enter the miles driven: 15
The miles / gallon for this tank was 1.5

Enter the gallons used ( -1 to quit ): -1

The overall average miles/gallon was 5.92857142857143
```

3.11 Write a program that receives as input a series of numbers. At the end of the input list, the program should output the total number of numbers input, the largest number, the smallest number, and the average of all the numbers.

ANS:

```
#!/usr/bin/perl
# Ex. 3.11: Ex03_11.pl:
# Inputting and evaluating numbers.

print( "Enter a number ( -1 to quit ): " );
chomp( $number = <STDIN> );

if ( $number != -1 ) {
   $smallest = $number;
   $largest = $number;
   $total = $number;
```

```
    $count++;

    print( "Enter a number ( -1 to quit ): " );
    chomp( $number = <STDIN> );
}

while ( $number != -1 ) {

    if ( $number < $smallest ) {
       $smallest = $number;
    }

    if ( $number > $largest ) {
       $largest = $number;
    }

    $total += $number;
    $count++;

    print( "Enter a number ( -1 to quit ): " );
    chomp( $number = <STDIN> );
}

if ( $count ) {
    print( "The total number of numbers was $count.\n" );
    print( "Largest number    $largest\n" );
    print( "Smallest number   $smallest\n" );
    print( "Average           ", $total / $count, "\n" );
}
```

```
Enter a number ( -1 to quit ): 3
Enter a number ( -1 to quit ): 4
Enter a number ( -1 to quit ): 5
Enter a number ( -1 to quit ): 6
Enter a number ( -1 to quit ): 7
Enter a number ( -1 to quit ): 8
Enter a number ( -1 to quit ): 9
Enter a number ( -1 to quit ): 0
Enter a number ( -1 to quit ): -1
The total number of numbers was 8.
Largest number    9
Smallest number   0
Average           5.25
```

3.12 [CD] A palindrome is a number or a text phrase that reads the same backwards as forwards. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a program that reads in a number of arbitrary length and checks if it is a palindrome. [*Hint:* Use the division and modulus operators to separate the number into its individual digits. Note that when you divide by 10, the remainder from this division will result in the last digit of the original number. For instance, 12345 divided by 10 results in a remainder of 5, and 789 divided by 10 results in a remainder of 9.]

ANS:

```
#!/usr/bin/perl
# Ex. 3.12: Ex03_12.pl
# Checking to see if a number is a palindrome.

print( "Enter a number: " );
chomp( $number = <STDIN> );

# For large numbers Perl may convert values to
# scientific notation. The if structure below
# is designed to handle this case.
if ( $number / 1000000000000000 > 1 ) {
   print( "The number is too large for this program.\n" );
}
else {
   $high = 10;

   while ( $number / $high > 1 ) {
      $high *= 10;
   }

   $low = 1;
   $high /= 10;

   while ( $high > $low ) {
      $digit1 = ( $number / $low ) % 10;
      $digit2 = ( $number / $high ) % 10;

      if ( $digit1 != $digit2 ) {
         $no = 1;
      }

      $low *= 10;
      $high /= 10;
   }

   if ( $no ) {
      print( "$number is not a palindrome.\n" );
   }
   else {
      print( "$number is a palindrome!\n" );
   }
}
```

```
Enter a number: 5555555
5555555 is a palindrome!
```

```
Enter a number: 565
565 is a palindrome!
```

```
Enter a number: 56766
56766 is not a palindrome.
```

3.13 [CD] Receive as input an integer containing only 0s and 1s (i.e., a "binary" integer), and print its decimal equivalent. [*Hint:* Use the modulus and division operators to pick off the "binary" number's digits one at a time from right to left. Just as in the decimal-number system, where the right most digit has a positional value of 1, the next digit to the left has a positional value of 10, then 100, then 1000, etc., in the binary-number system, the right most digit has a positional value of 1, the next digit to the left has a positional value of 2, then 4, then 8, etc. Thus the decimal number 234 can be interpreted as 4 * 1 + 3 * 10 + 2 * 100. The decimal equivalent of binary 1101 is 1 * 1 + 0 * 2 + 1 * 4 + 1 * 8 or 1 + 0 + 4 + 8, or 13.]

ANS:

```
#!/usr/bin/perl
# Ex. 3.13: Ex03_13.pl
# Converting from binary to decimal.

print( "Input a binary number: " );
chomp( $binary = <STDIN> );

$bit = 0;
$decimal = 0;

while ( $binary / ( 10 ** $bit ) >= 1 ) {
   $digit = ( $binary / ( 10 ** $bit ) ) % 10;
   $decimal += $digit * 2 ** $bit;
   $bit++;
}

print( "The decimal representation for $binary is $decimal.\n" );
```

```
Input a binary number: 101
The decimal representation for 101 is 5.
```

```
Input a binary number: 1000
The decimal representation for 1000 is 8.
```

3.14 The factorial of a nonnegative integer n is written as $n!$ (pronounced "n factorial") and is defined as follows:

$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$ (for values of n greater than or equal to 1)

and

$n! = 1$ (for $n = 0$).

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120.

a) Write a program that reads a nonnegative integer and computes and prints its factorial.

ANS:

```
#!/usr/bin/perl
# Ex. 3.14a: Ex03_14a.pl
# Calculating factorials.

print( "Enter a number: " );
chomp( $number = <STDIN> );

$factorial = 1;

$i = $number;

while ( $i > 0 ) {
   $factorial *= $i;
   $i--;
}

print( "$number! = $factorial\n" );
```

```
Enter a number: 5
5! = 120
```

b) Write a program that estimates the value of the mathematical constant *e* by using the following formula: (One way to do this is to have the program print the result after each term is added, so you can see what value *e* comes close to. Prompt the user for the number of terms they wish to have calculated, to ensure that the program does not go on forever.)

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

ANS:

```
#!/usr/bin/perl
# Ex. 3.14b: Ex03_14b.pl
# Calculating the value of e.

print( "How many iterations should we use to compute e? " );
chomp( $iterations = <STDIN> );

$i = 1;
$e = 1;

while ( $i <= $iterations ) {
   $factorial = 1;
   $j = $i;

   while ( $j ) {
      $factorial *= $j;
      $j--;
   }

   $e += 1 / $factorial;
```

```
    print( "The value of e at iteration number $i is $e\n" );
    $i++;
}

print( "With $iterations iterations, e = $e\n" );
```

```
How many iterations should we use to compute e? 10
The value of e at iteration number 1 is 2
The value of e at iteration number 2 is 2.5
The value of e at iteration number 3 is 2.66666666666667
The value of e at iteration number 4 is 2.70833333333333
The value of e at iteration number 5 is 2.71666666666667
The value of e at iteration number 6 is 2.71805555555556
The value of e at iteration number 7 is 2.71825396825397
The value of e at iteration number 8 is 2.71827876984127
The value of e at iteration number 9 is 2.71828152557319
The value of e at iteration number 10 is 2.71828180114638
With 10 iterations, e = 2.71828180114638
```

c) Write a program that computes the value of e^x by using the following formula (again, have the program print after each iteration, stopping after a specified number of terms) and prints the current value after each iteration:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

ANS:

```
#!/usr/bin/perl
# Ex. 3.14c: Ex03_14c.pl
# Calculating e to a specific power.

print( "What power should e be raised to? " );
chomp( $power = <STDIN> );
print( "How many iterations should we use to compute e? " );
chomp( $iterations = <STDIN> );

$i = 1;
$e = 1;

while ( $i <= $iterations ) {
   $factorial = 1;
   $j = $i;

   while ( $j ) {
      $factorial *= $j;
      $j--;
   }

   $e += ( $power ** $i ) / $factorial;
   print( "At iteration number $i, e^$power = $e\n" );
   $i++;
}
```